AMATH 482 Homework 4

Wenxuan Liu

March 10, 2021

Abstract

We have a large set of hand written digits from zero to nine stored as images. The images are separated into training and testing sets, and our goal is to train classifiers that correctly identify which digit is written on each images. The training set is used for training classifier and the testing set is used to test accuracy. There are three kinds of classifier we are building: Linear Discriminant Analysis (LDA), SVM (support vector machines), and decision tree. We are going to build LDA that classify two digits and another LDA classify three digits, and we identify the pair of digits that is hardest/easiest to distinguish. With the hardest/easiest digit pairs, we are going to build SVM and decision tree classifiers that distinguish between all ten digits, then compare LDA, SVM, and decision tree classifiers on those hardest/easiest digit pairs.

1 Introduction and Overview

We make LDA, SVD, and decision tree classifiers on ten digits and compare their performance in the following four steps:

1. Load data and make a matrix of all training images. Do SVD on the image matrix and project the images onto principal components. Plot the projection on principal components and see how well the clusters are separated. Lastly, reconstruct the images using low rank approximation to determine how may features to keep in later steps.

2. Create 2 or 3 digits LDA classifier using the feature number and low rank approximation from last step. To get LDA seperate 2 digits, we look for the feature that best separate two digits and compute the threshold accordingly. To get LDA separate 3 digits, we use two LDA for 2 digits to narrow down the result to one digit. Compute accuracy for all digit pairs to get easiest or hardest pair to distinguish.

3. Create SVD and decision tree classifier using MatLab functions fitcecoc and fitctree.

4. Compare LDA, SVD, and decision tree performance on the easiest and hardest digit pairs.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD) & Principal Component Analysis (PCA)

As we learned from our text book (lecture 11) [3], The singular value decomposition of the matrix A is:

$$A = U\Sigma V^* \tag{1}$$

Where where $U \in \mathbf{R}^{m\times m}$ and $\Sigma \in \mathbf{R}^{n\times n}$ are unitary matrices, and $V \in \mathbf{R}^{m\times n}$ is diagonal. Based on this idea, multiply by A on the left can be decomposed to: rotate by V^* to align V_n with the axes, stretch by Σ in each direction, and rotate back by U.
We can construct low rank approximation for matrices using SVD: We can tell the most important principal components in a matrix and pick the most variation out of the matrix and reduce redundancy.

2.2 Low-Dimensional Approximations

From text book (lecture 11) [3], we learned Low-Dimensional Approximations of matrix, which is an application of SVD. Given a matrix A with rank r, the best rank N approximation of A is given by:

$$A_N = \sum_{j-1}^{N} \sigma_j u_j v_j^* \tag{2}$$

2.3 Linear Discriminant Analysis

From text book (lecture 19) [3], we learned Linear Discriminant Analysis. The goal of LDA is two-fold: find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data.we calculate the means for each of our groups for each feature (μ_1, μ_2). We can then define the between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{3}$$

Define within-class scatter matrix to be:

$$S_w = \sum_{j=1}^{2} \sum_{X} (X - \mu_j)(X - \mu_j)^T \tag{4}$$

The goal is then to find a vector w such that

$$W = argmax \frac{W^T S_B W}{W^T S_w W} \tag{5}$$

Lastly, we simplify the above equation to find the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem:

$$S_B W = \lambda S_w W \tag{6}$$

2.4 Decision Tree

Decision trees can be used to visually and explicitly represent decisions and decision making in data science. The Decision tree is a binary tree with decisions as each nodes, and true/false value in each node lead to the next node(decision) until a leaf node is reached as a prediction[2]. Decision tree is a common type of classifier.

2.5 SVM

Support vector machine is an algorithm to find the N-D hyper-plane that best separate the data points. Although there are many hyper-planes that separate the data, SVM is the one with maximized margin between data groups being separated[1]. SVM is a common way of building a classifier.

3 Algorithm Implementation and Development

3.1 Process Images

1. First we load the images into MatLab. Then, we resize each 28×28 image to a column vector. Then we get a matrix with each column as an image, and we do SVD on the matrix after demeaning (subtract mean from columns).

2. Plot all singular values and on another plot, project the matrix of image onto the first 2,3,4 vectors. In our plot, we color each digit using different colors in order to see how digits are grouped.

3. Reconstruct the image using low rand approximations and find the number of singular values we need to preserve most of the image details.

3.2 Build Linear Classifiers

1. We want to find the feature W that best describe the difference between two digits, so we use the SVD and feature number from above to calculate the best vector W and threshold. Sudo code:

Algorithm 1: Extract Can Position Algorithm

```
Pull out the first digit and the first k features
Pull out the second digit and the first k features
for j = 1 : [first digit] do
  calculate S_W
end for
for j = 1 : [second digit] do
  calculate S_W
end for
calculate S_B
solve the eigenvalue problem to get W
project the first digit onto W and sort
project the second digit onto W and sort
while sortOne(t1) > sortTwo(t2) do
  t1 = t1 - 1
  t2 = t2 - 2
end while
threshold = (sortOne(t1) + sortZero(t2))/2
```

2. For all pairs of digits, we create a classifier in previous step and test on the test set. We create two for-loops to generate all digit pairs and for each one of them, train linear model and test on test set. Then we record the pairs with maximum or minimum accuracy.

3. For three digits linear classifier, we build two 2-digit classifier: 1. classify digit 1 or not; 2. classify digit 2 or digit 3. Build 3-digit classifier for all 3-digit groups and find least and largest accuracy.

3.3 Decision Tree and SVD

We first load training and testing data and their labels. Create SVD and decision tree classifier using MatLab functions fitcecoc and fitctree. Then we use the trained models to predict the test set and get the accuracy. See Appendix B for MatLab code.

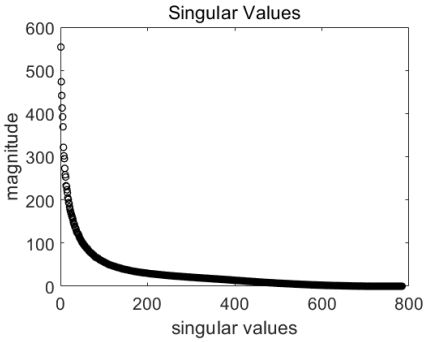
3.4 Compare three classifiers

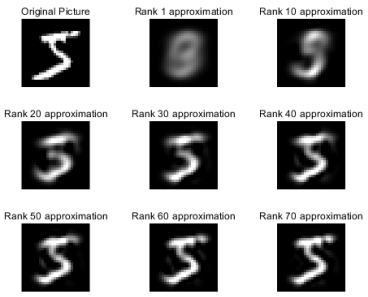
We don't need to run LDA again because LDA told us originally the easiest and hardest digits to distinguish (and accuracies). We train decision tree and SVD on the two pairs of digits and get the training/testing accuracies. We compare the three methods to conclude which one is at those two digit pairs.

4 Computational Results

4.1 Processing images

The singular values after demeaning image data shows that there is no dominating principal components. The low rank approximation tells us that the first 50 principal components (features) provide enough information to recognize the image, so we will build linear classifiers using first 50 features. See figure below:





(b) Low Rank Approximations

Figure 1: PCA Analysis

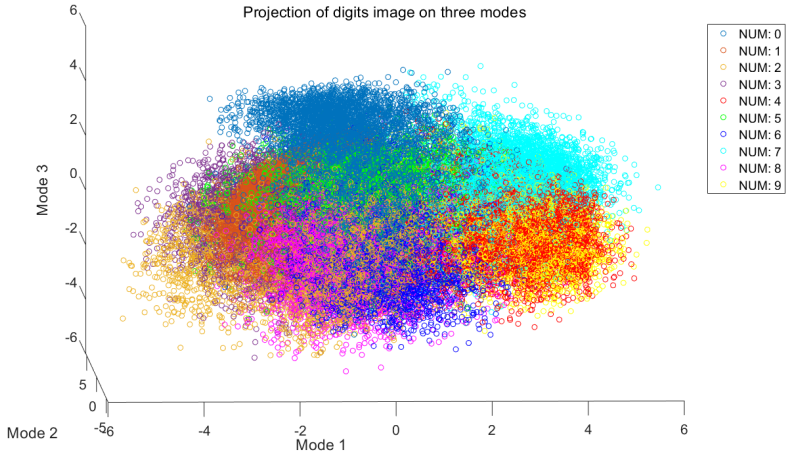
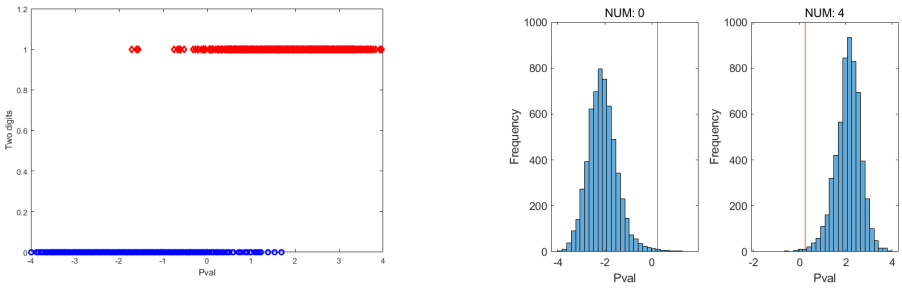


Figure 2: Cluster of Digits

4.2 Build Linear Classifier

Train and test on 2-digit data:

We train linear classifiers and predict on test sets for all digit pairs.
The easiest distinguishable pair of digit is: (0,4), and test accuracy is 0.99847; training accuracy is 0.9949.

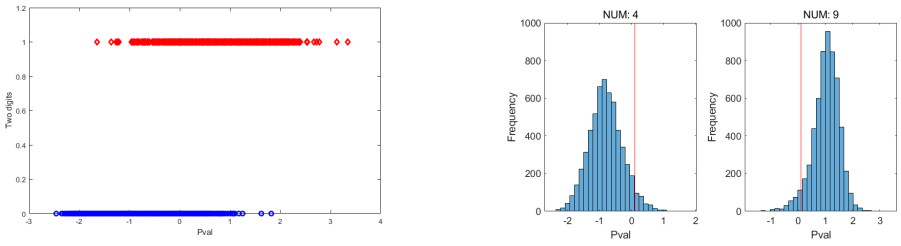


(a) Projection on W

(b) Histogram and threshold

Figure 3: Linear Classifier for digits 0 an 4

The hardiest distinguishable pair of digit is: (4,9), and test accuracy is 0.94726; train accuracy is 0.9553.



(a) Projection on W (b) Projection on CPs

Figure 4: Linear Classifier for digits 4 and 9

Repeat for all of 3-digit data sets:
The easiest distinguishable set of 3 digits is: (0,1,4), and test accuracy is 0.99677.
The hardiest distinguishable set of 3 digits is: (3,5,8), and accuracy is 0.89082.

4.3 Decision Tree and SVM

1. Decision tree:
   Train accuracy: 0.8777
   Test accuracy: 0.9679

2. SVM:
   Train accuracy: 0.973666666666667
   Test accuracy: 0.9438

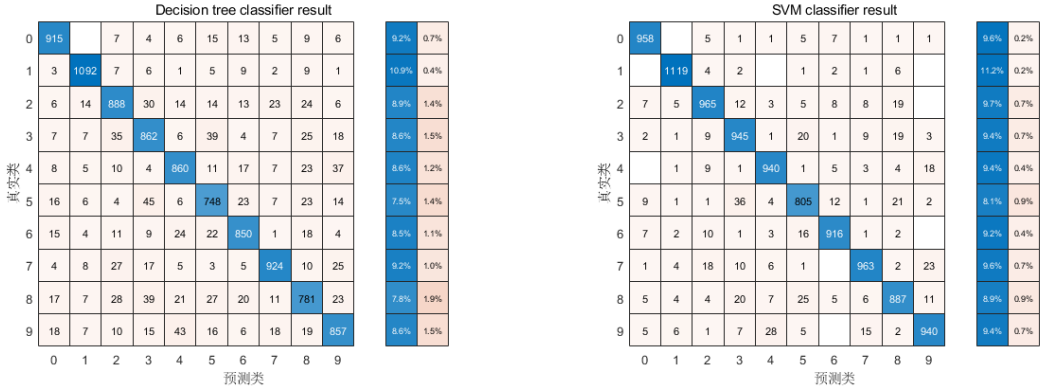


Figure 5: Accuracy Analysis

4.4 Compare Three Classifiers

The performance of three classifiers on digits 0, 4 (Easiest pair):

Accuracy	Linear	Decision Tree	SVM
Train	0.994900127496813	0.998045048873778	1
Test	0.998470948012233	0.988786952089704	0.994903160040775

The performance of three classifiers on digits 4, 9 (Hardest pair):

Accuracy	Linear	Decision Tree	SVM
Train	0.955304893562887	0.993893647697396	0.980493596811127
Test	0.947262682069312	0.952285283776997	0.970868910095429

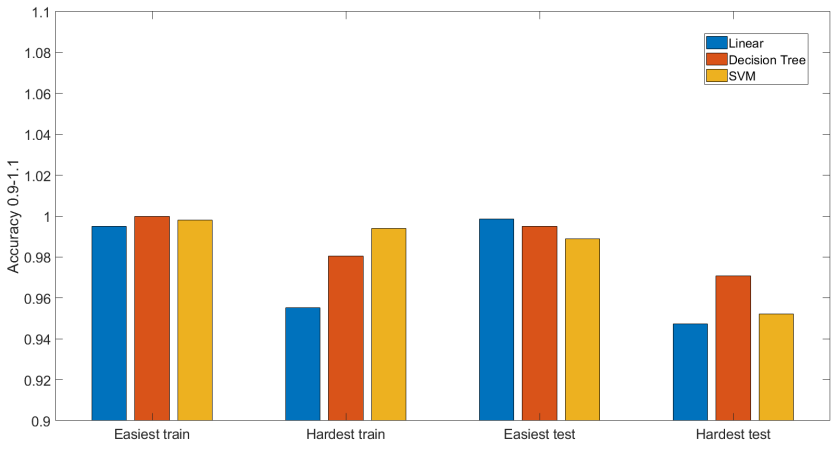


Figure 6: Comparison over two pairs, three classifiers, and test/train

5 Summary and Conclusions

We found the easiest and hardest 2 digits and 3 digits to classify. The result is very reasonable: 0 is always just a loop, while 4 is a loop with two tails, so they are very easy to distinguish. 4 and 9 looks most similar through human eyes, so the classifiers also think it is the hardest.
For comparison part, we created 2-digit and 3-digit linear classifiers, 10-digit and 2-digit SVM, decision tree classifiers. All of them shows high accuracy on both train and test data sets. One conclusion we can draw from our data is that train accuracy is generally higher than test accuracy. Another conclusion is that there is no clear winner in the three classifier for 2-digit pairs.

References

[1] Rohith Gandhi. *Support vector machine - introduction to machine learning algorithms*. July 2018. URL: `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`.

[2] Prashant Gupta. *Decision Trees in Machine Learning*. Nov. 2017. URL: `https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052`.

[3] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal.
- `imshow(I)` displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- `C = bsxfun(fun,A,B)` applies the element-wise binary operation specified by the function handle fun to arrays A and B.
- `B = reshape(A,sz)` reshapes A using the size vector, sz, to define size(B). For example, reshape(A,[2,3]) reshapes A into a 2-by-3 matrix. sz must contain at least 2 elements, and prod(sz) must be the same as numel(A).
- `I2 = im2double(I)` converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1].
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
  If X is a vector, then find returns a vector with the same orientation as X.
  If X is a multidimensional array, then find returns a column vector of the linear indices of the result.
- `plot3(X,Y,Z)` plots coordinates in 3-D space.
  To plot a set of coordinates connected by line segments, specify X, Y, and Z as vectors of the same length.
  To plot multiple sets of coordinates on the same set of axes, specify at least one of X, Y, or Z as a matrix and the others as vectors.
- `tree = fitctree(Tbl,ResponseVarName)` returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes) contained in the table Tbl and output (response or labels) contained in Tbl.ResponseVarName. The returned binary tree splits branching nodes based on the values of a column of Tbl.
- `[labelIdx,score] = predict(categoryClassifier,I)` returns the predicted label index and score for the input image.
- `Mdl = fitcecoc(Tbl,ResponseVarName)` returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses K(K – 1)/2 binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). Mdl is a ClassificationECOC model.
- `S = sum(A)` returns the sum of the elements of A along the first array dimension whose size does not equal 1.
  If A is a vector, then sum(A) returns the sum of the elements.
  If A is a matrix, then sum(A) returns a row vector containing the sum of each column.
  If A is a multidimensional array, then sum(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors. This dimension becomes 1 while the sizes of all other dimensions remain the same.
- `B = categorical(A)` creates a categorical array from the array A. The categories of B are the sorted unique values from A.
- `B = reordercats(A)` reorders the categories in the categorical array, A, to be in alphanumeric order.
- `bar(y)` creates a bar graph with one bar for each element in y. If y is an m-by-n matrix, then bar creates m groups of n bars.
- `e = eig(A)` returns a column vector containing the eigenvalues of square matrix A.

- `n = norm(v,p)` returns the generalized vector p-norm.
- `B = sort(A)` sorts the elements of A in ascending order.
  If A is a vector, then sort(A) sorts the vector elements.
  If A is a matrix, then sort(A) treats the columns of A as vectors and sorts each column.
  If A is a multidimensional array, then sort(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors.
- `histogram(X)`creates a histogram plot of X. The histogram function uses an automatic binning algorithm that returns bins with a uniform width, chosen to cover the range of elements in X and reveal the underlying shape of the distribution. histogram displays the bins as rectangles such that the height of each rectangle indicates the number of elements in the bin.
- `Mdl = fitclinear(X,Y)`returns a trained linear classification model object Mdl that contains the results of fitting a binary support vector machine to the predictors X and class labels Y.

Appendix B MATLAB Code

MATLAB code that produce the outcomes discussed in this paper.

```
%% classify 2 digits
clear all; close all; clc;

[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
images = im2double(images);
imgSize = size(images);
images = reshape(images,[imgSize(1) * imgSize(2), imgSize(3)]);
images = bsxfun(@minus, images, mean(images,2));
[U,S,V] = svd(images,'econ');

figure(1)
plot(diag(S),'ko','Linewidth',1)
set(gca,'Fontsize',16)
title('Singular Values')
xlabel('singular values')
ylabel('magnitude')

figure(2)
Y = U' * images;
colors = [[0 0.4470 0.7410];[0.8500 0.3250 0.0980];[0.9290 0.6940 0.1250];[0.4940 0.1840
    0.5560];[1 0 0];[0 1 0];[0 0 1];[0 1 1];[1 0 1];[1 1 0]];
for j = 0:9
    idx = find(labels==j);
    plot3(Y(2,idx), Y(3,idx), Y(5,idx), 'o', 'Color', colors(j+1,:), 'displayname', sprintf('NUM:
        %s',string(j))), hold on
end
set(gca,'Fontsize',16)
legend()
title('Projection of digits image on three modes')
xlabel('Mode 1')
ylabel('Mode 2')
zlabel('Mode 3')

%%
clear all; close all; clc;

[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
images = im2double(images);
```

```
imgSize = size(images);
images = reshape(images,[imgSize(1) * imgSize(2), imgSize(3)]);
[U,S,V] = svd(images,'econ');

original = reshape(images, imgSize);
appro1 = reshape(U(:,1)*S(1,1)*V(:,1)', imgSize);
appro10 = reshape(U(:,1:10)*S(1:10,1:10)*V(:,1:10)', imgSize);
appro20 = reshape(U(:,1:20)*S(1:20,1:20)*V(:,1:20)', imgSize);
appro30 = reshape(U(:,1:30)*S(1:30,1:30)*V(:,1:30)', imgSize);
appro40 = reshape(U(:,1:40)*S(1:40,1:40)*V(:,1:40)', imgSize);
appro50 = reshape(U(:,1:50)*S(1:50,1:50)*V(:,1:50)', imgSize);
appro60 = reshape(U(:,1:60)*S(1:60,1:60)*V(:,1:60)', imgSize);
appro70 = reshape(U(:,1:70)*S(1:70,1:70)*V(:,1:70)', imgSize);
subplot(3,3,1), imshow(original(:, :, 1));
title('Original Picture')
subplot(3,3,2), imshow(appro1(:, :, 1));
title('Rank 1 approximation')
subplot(3,3,3), imshow(appro10(:, :, 1));
title('Rank 10 approximation')
subplot(3,3,4), imshow(appro20(:, :, 1));
title('Rank 20 approximation')
subplot(3,3,5), imshow(appro30(:, :, 1));
title('Rank 30 approximation')
subplot(3,3,6), imshow(appro40(:, :, 1));
title('Rank 40 approximation')
subplot(3,3,7), imshow(appro50(:, :, 1));
title('Rank 50 approximation')
subplot(3,3,8), imshow(appro60(:, :, 1));
title('Rank 60 approximation')
subplot(3,3,9), imshow(appro70(:, :, 1));
title('Rank 70 approximation')
%%
clear all; close all; clc;

[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
images = im2double(images);
imgSize = size(images);
images = reshape(images,[imgSize(1) * imgSize(2), imgSize(3)]);
[U,S,V] = svd(images,'econ');
feature = 50;
Y = U' * images;
accuracy = [];
for num1 = 0:9
    for num2 = num1 + 1 : 9
        [threshold,w,sortdog,sortcat] = digits_trainer_2(num1, num2, feature, Y, labels);
        accuracy = [accuracy; [num1 num2 checkAccuracy(num1, num2, feature, U, threshold, w)
            checkTrainAccuracy(num1, num2, feature, U, threshold, w)]];
    end
end
hardest = accuracy(accuracy(:,3) == min(accuracy(:,3)),:); % [4,9,0.947262682069312]
easiest = accuracy(accuracy(:,3) == max(accuracy(:,3)),:); % [0,4,0.998470948012233]

%% classify three digits
clear all; close all; clc;

accuracy = [];
for num1 = 0:9
    for num2 = num1 + 1 : 9
        for num3 = num2 + 1 : 9
```

```
            accuracy = [accuracy; [num1 num2 num3 classify_3(num1, num2, num3)]];
        end
    end
end
hardest = accuracy(accuracy(:,4) == min(accuracy(:,4)),:); % [3,5,8,0.890820584144645]
easiest = accuracy(accuracy(:,4) == max(accuracy(:,4)),:); % [0,1,4,0.996771068776235]

%% using decision tree
clear all; close all; clc;

[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
images = im2double(images);
imgSize = size(images);
images = reshape(images,[imgSize(1) * imgSize(2), imgSize(3)]);
[testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
testImages = im2double(testImages);
testImagesSize = size(testImages);
testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);

tree=fitctree(images',labels);
test_labels = predict(tree,testImages');
train_labels = predict(tree,images');
train_accuracy = sum(labels==train_labels)/length(train_labels);
test_accuracy = sum(testLabels==test_labels)/length(test_labels);
view(tree,'Mode','graph');
save('DT.mat','tree');
%% using SVM
clear all; close all; clc;

[images, labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
images = im2double(images);
imgSize = size(images);
images = reshape(images,[imgSize(1) * imgSize(2), imgSize(3)]);

[testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
testImages = im2double(testImages);
testImagesSize = size(testImages);
testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);

Mdl = fitcecoc(images',labels);

train_labels = predict(Mdl, images');
test_labels = predict(Mdl,testImages');

train_accuracy = sum(labels==train_labels)/length(train_labels);
test_accuracy = sum(testLabels==test_labels)/length(test_labels);
save('SVM.mat','Mdl');
%% continued
load('SVM.mat')
load('DT.mat')
[train_images, train_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
train_images = im2double(train_images);
imgSize = size(train_images);
train_images = reshape(train_images,[imgSize(1) * imgSize(2), imgSize(3)]);

[testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
testImages = im2double(testImages);
testImagesSize = size(testImages);
testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);
```

```
SVMlabels = predict(Mdl,testImages');
treeLabels = predict(tree,testImages');
confusionchart(testLabels,SVMlabels,'RowSummary','total-normalized');
title('SVM classifier result')
figure()
confusionchart(testLabels,treeLabels,'RowSummary','total-normalized');
title('Decision tree classifier result')
%% compare LDA SVM and decision tree
clear all; close all; clc;
num1 = 0;
num2 = 4;

[train_images, train_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
train_images = im2double(train_images);
imgSize = size(train_images);
train_images = reshape(train_images,[imgSize(1) * imgSize(2), imgSize(3)]);

[testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
testImages = im2double(testImages);
testImagesSize = size(testImages);
testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);

selectedTrainImages = [train_images(:,train_labels==num1) train_images(:,train_labels==num2)];
selectedTrainLabels = [train_labels(train_labels==num1); train_labels(train_labels==num2)];

selectedTestImages = [testImages(:,testLabels==num1) testImages(:,testLabels==num2)];
selectedTestLabels = [testLabels(testLabels==num1); testLabels(testLabels==num2)];

% LDA already done
% [4,9,0.947262682069312]
% [0,4,0.998470948012233]

% SVM on 0,4
Mdl = fitcecoc(selectedTrainImages',selectedTrainLabels);
test_labels = predict(Mdl,selectedTestImages');
trainlabels = predict(Mdl,selectedTrainImages');
easiestSVMaccuracyTrain = sum(selectedTrainLabels==trainlabels)/length(trainlabels);
easiestSVMaccuracyTest = sum(selectedTestLabels==test_labels)/length(test_labels);
% decision tree on 0,4
tree=fitctree(selectedTrainImages',selectedTrainLabels);
test_labels = predict(tree,selectedTestImages');
trainlabels = predict(tree,selectedTrainImages');
easiestDTaccuracyTrain = sum(selectedTrainLabels==trainlabels)/length(trainlabels);
easiestDTaccuracyTest = sum(selectedTestLabels==test_labels)/length(test_labels);

num1 = 4;
num2 = 9;

[train_images, train_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
train_images = im2double(train_images);
imgSize = size(train_images);
train_images = reshape(train_images,[imgSize(1) * imgSize(2), imgSize(3)]);

[testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
testImages = im2double(testImages);
testImagesSize = size(testImages);
testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);
```

```
selectedTrainImages = [train_images(:,train_labels==num1) train_images(:,train_labels==num2)];
selectedTrainLabels = [train_labels(train_labels==num1); train_labels(train_labels==num2)];

selectedTestImages = [testImages(:,testLabels==num1) testImages(:,testLabels==num2)];
selectedTestLabels = [testLabels(testLabels==num1); testLabels(testLabels==num2)];

% LDA already done
% [4,9,0.947262682069312]
% [0,4,0.998470948012233]

% SVM on 4,9
Mdl = fitcecoc(selectedTrainImages',selectedTrainLabels);
test_labels = predict(Mdl,selectedTestImages');
trainlabels = predict(Mdl,selectedTrainImages');
hardestSVMaccuracyTrain = sum(selectedTrainLabels==trainlabels)/length(trainlabels);
hardestSVMaccuracyTest = sum(selectedTestLabels==test_labels)/length(test_labels);
% decision tree on 4,9
tree=fitctree(selectedTrainImages',selectedTrainLabels);
test_labels = predict(tree,selectedTestImages');
trainlabels = predict(tree,selectedTrainImages');
hardestDTaccuracyTrain = sum(selectedTrainLabels==trainlabels)/length(trainlabels);
hardestDTaccuracyTest = sum(selectedTestLabels==test_labels)/length(test_labels);
%%
X = categorical({'Easiest train','Hardest train', 'Easiest test','Hardest test'});
X = reordercats(X,{'Easiest train','Hardest train', 'Easiest test','Hardest test'});
Y = [0.994900127496813 easiestSVMaccuracyTrain easiestDTaccuracyTrain;
    0.955304893562887 hardestSVMaccuracyTrain hardestDTaccuracyTrain;
    0.998470948012233 easiestSVMaccuracyTest easiestDTaccuracyTest;
    0.947262682069312 hardestSVMaccuracyTest hardestDTaccuracyTest];
h = bar(X,Y)
ylim([0.9 1.1])
ylabel('Accuracy 0.9-1.1')
set(h, {'DisplayName'}, {'Linear','Decision Tree','SVM'}')
legend()
set(gca,'Fontsize',20)
%%
function [threshold,w,sortOne,sortZero] = digits_trainer_2(num1, num2, feature, Y, labels)
    draw = num1==4 & num2 ==9;
    one = Y(1:feature,labels==num1);
    zero = Y(1:feature,labels==num2);
    mo = mean(one,2);
    mz = mean(zero,2);
    [~, no] = size(one);
    [~, nz] = size(zero);
    Sw = 0; % within class variances
    for k = 1:no
        Sw = Sw + (one(:,k) - mo)*(one(:,k) - mo)';
    end
    for k = 1:nz
       Sw = Sw + (zero(:,k) - mz)*(zero(:,k) - mz)';
    end

    Sb = (mo-mz)*(mo-mz)'; % between class

    [V2, D] = eig(Sb,Sw); % linear disciminant analysis
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
```

```
    vOne = w'*one;
    vZero = w'*zero;

    if mean(vOne) > mean(vZero)
        w = -w;
        vOne = -vOne;
        vZero = -vZero;
    end

    sortOne = sort(vOne);
    sortZero = sort(vZero);

    t1 = length(sortOne);
    t2 = 1;
    while sortOne(t1) > sortZero(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end
    threshold = (sortOne(t1) + sortZero(t2))/2;

    if draw
        figure(4)
        plot(vOne,zeros(no),'ob','Linewidth',2)
        hold on
        plot(vZero,ones(nz),'dr','Linewidth',2)
        ylim([0 1.2])
        xlabel('Pval')
        ylabel('Two digits')

        figure(5)
        subplot(1,2,1)
        histogram(sortOne,30); hold on, plot([threshold threshold], [0 1000], 'r')
        set(gca,'Fontsize',14)
        title(sprintf('NUM: %s',string(num1)))
        xlabel('Pval')
        ylabel('Frequency')
        subplot(1,2,2)
        histogram(sortZero,30); hold on, plot([threshold threshold], [0 1000], 'r')
        set(gca,'Fontsize',14)
        title(sprintf('NUM: %s',string(num2)))
        xlabel('Pval')
        ylabel('Frequency')
    end
end

function accuracy = classify_3(num1, num2, num3)
    [train_images, train_labels] = mnist_parse('train-images.idx3-ubyte',
        'train-labels.idx1-ubyte');
    train_images = im2double(train_images);
    imgSize = size(train_images);
    train_images = reshape(train_images,[imgSize(1) * imgSize(2), imgSize(3)]);

    [testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
    testImages = im2double(testImages);
    testImagesSize = size(testImages);
    testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);

    selectedImages1 = [train_images(:,train_labels==num1) train_images(:,train_labels==num2)
        train_images(:,train_labels==num3)];
```

```
    num1OrNotLabels = [train_labels(train_labels==num1); -ones(sum(train_labels==num2),1);
        -ones(sum(train_labels==num3),1)];

    selectedImages2 = [train_images(:,train_labels==num2) train_images(:,train_labels==num3)];
    num2OrNotLabels = [train_labels(train_labels==num2); train_labels(train_labels==num3)];

    Mdl1 = fitclinear(selectedImages1',num1OrNotLabels);
    Mdl2 = fitclinear(selectedImages2',num2OrNotLabels);

    selectedTestImages = [testImages(:,testLabels==num1) testImages(:,testLabels==num2)
        testImages(:,testLabels==num3)];
    selectedTestLabels = [testLabels(testLabels==num1); testLabels(testLabels==num2);
        testLabels(testLabels==num3)];
    test_labels1 = predict(Mdl1,selectedTestImages');
    test_labels2 = predict(Mdl2,selectedTestImages');
    for j = 1:length(test_labels1)
        if test_labels1(j) < 0
            test_labels1(j) = test_labels2(j);
        end
    end

    accuracy = sum(selectedTestLabels==test_labels1)/length(test_labels1);
end

function accuracy = checkAccuracy(num1, num2, feature, U, threshold, w)
    [testImages, testLabels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
    testImages = im2double(testImages);
    testImagesSize = size(testImages);
    testImages = reshape(testImages,[testImagesSize(1) * testImagesSize(2), testImagesSize(3)]);

    selectedImages = [testImages(:,testLabels==num1) testImages(:,testLabels==num2)];
    selectedLabels = [testLabels(testLabels==num1); testLabels(testLabels==num2)];
    U = U(:,1:feature); % Add this in
    IMat = U'*selectedImages; % PCA projection
    pval = w'*IMat;

    result = [];
    for n = 1 : length(pval)
        if pval(n) > threshold
            result = [result num2];
        else
            result = [result num1];
        end
    end
    accuracy = sum(selectedLabels'==result)/length(result);

end

function accuracy = checkTrainAccuracy(num1, num2, feature, U, threshold, w)
    [train_images, train_labels] = mnist_parse('train-images.idx3-ubyte',
        'train-labels.idx1-ubyte');
    train_images = im2double(train_images);
    imgSize = size(train_images);
    train_images = reshape(train_images,[imgSize(1) * imgSize(2), imgSize(3)]);

    selectedImages = [train_images(:,train_labels==num1) train_images(:,train_labels==num2)];
    selectedLabels = [train_labels(train_labels==num1); train_labels(train_labels==num2)];
    U = U(:,1:feature); % Add this in
    IMat = U'*selectedImages; % PCA projection
```

```
    pval = w'*IMat;

    result = [];
    for n = 1 : length(pval)
        if pval(n) > threshold
            result = [result num2];
        else
            result = [result num1];
        end
    end
    accuracy = sum(selectedLabels'==result)/length(result);

end
```